Use Multi-Agent Reinforcement Learning to Find the Best Charging Stations for Trucks

EE 556 Course Project

Number of student authors: 1

Yijing Jiang, yijingji@usc.edu

05/05/2021

1. Introduction and Motivation

1.1. Introduction

The multi agent reinforcement learning (MARL) system focuses on the situation where there are more than one agents in the environment. In the MARL system, there will be a reward to be optimized and some policies to be updated. We are curious about how the agents will communicate with each other, how the actions of agents taken will affect others and so on. Some of the recent work is using the combination of reinforcement learning and deep learning. Deep learning works well on solving high dimensional problems, which allows us to apply the MARL system to more realistic situations. We will first introduce some of the concepts in the MARL system and clarify which problem fields we will focus on. And then we will try different algorithms to further understand how they work together.

1.2. Motivation

Many situations could be simplified as MARL problems. Here is one: some big companies have their own transportations and are using electric vehicles (EVs) recently. However, as the charging stations for EVs are not as popular as gas stations, finding available chargers for the EVs is a problem. Also, there are some matching problems. For example, the type of plugs may not match the type of EVs, and the capacity power of plugs may also not match the capacity power of EVs. Thus, we need to find how to schedule all the trucks to proper chargers, which reminds me of what we have learned in this course.

2. Background and Problem Statement

2.1. Background

2.1.1. Notation of MARL:

Notation of MARL is similar to the single agent system. There will be n agents marked as $a \in A = \{1, 2, 3, ..., n\}$. The state of the environment is $s \in S$. Each agent could take an action $u^a \in U$, which will form the joint action $\mathbf{u} \in \mathbf{U} = U^n$. The transition probability will be related to the current state, current actions and the next state: $P(s' \mid s, \mathbf{u})$. The reward will also be affected by the current state, current actions and the next state: $r(s, \mathbf{u}, s')$. We always assume that the transition and the reward functions are unknown for the agents, but they will be defined in the environment in advance.

2.1.2. Problem Type:

Consider the communication between the agents, there will be several types of situations. In different types, the format of rewards will be different too.
One is **cooperative** with the same rewards at the most time. The second is **competitive** with zero sum rewards, which means the agents will receive rewards if they win the game and be punished if they lose the game. Some situations will include **both cooperative and competitive**, like soccer games.

The rewards could have more values, which is called general sum.

2.1.3. Nash Equilibrium:

The convergence of MARL is difficult to define. Because if one agent changes its action, all the other agents' choices may be affected. Therefore, we define the Nash equilibrium for MARL system as following:

Assume $\boldsymbol{\pi}^*$ is the final optimal policy with which all the agents could reach the best total reward; $\boldsymbol{\pi}^a$ is the policy of agent a; $\boldsymbol{\pi}^{*a}$ is the optimal policy of agent a; $\boldsymbol{\pi}^{*-a}$ is the optimal policy of other agents. Then the objective function is maximized based on $\boldsymbol{\pi}^{*-a}$:

$$J^a(\boldsymbol{\pi}^{*a}, \boldsymbol{\pi}^{*-a}) \geq J^a(\boldsymbol{\pi}^a, \boldsymbol{\pi}^{*-a}),\ \forall a,\ \forall \boldsymbol{\pi}^a$$

which means while all the other agents' policy remain the same, the agent policy could not get better than $\boldsymbol{\pi}^{*a}$.

2.1.4. Train and Execution Structure:

How the agents communicate in the training and executing will also affect whether they could reach Nash equilibrium quickly and safely. One structure is that all the agents do not communicate with each other, which is called **fully decentralized** structure. It is like several single agents optimizing their policies separately. They only know their own states and rewards, and take action according to their own policy. There is also one structure called **fully centralized**, where agents will know others' position and rewards. We train the joint policy by using the total rewards. In execution, we need to know the observation of all the agents to decide the joint action. The last one is **centralised training and decentralized execution**. We will use all the information to train the separate policies.

2.1.5. Iteration Algorithms:

We could update the iteration based on different functions. There will be three different methods: **value based** algorithm, **policy based** algorithm and **actor-critic** algorithm. In this report we will mainly focus on the **value based** iteration algorithms. Firstly, we will use the **action value function** (Q learning) to implement the algorithms. Secondly, we will try **Deep Q Network** (DQN) to find the optimal policy.

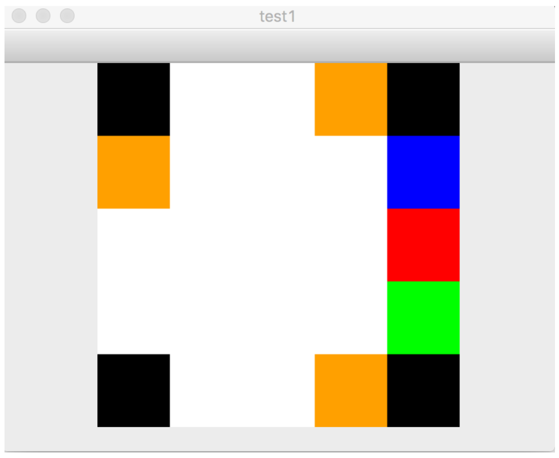
2.2. Problem Statement

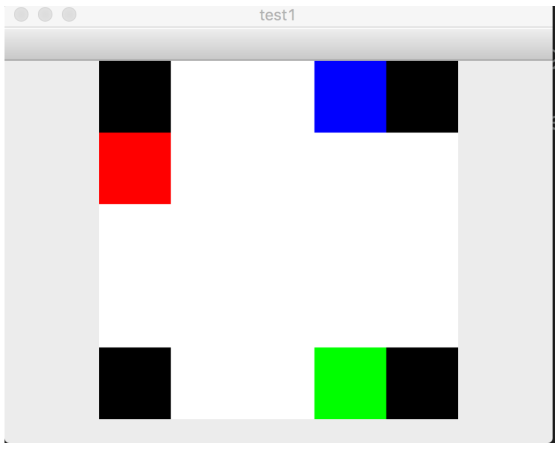
In conclusion, for the simplified problem of finding the best charging stations for trucks, we will firstly implement an MARL system which is **cooperative** and **fully observed**. Secondly, we will try to use **Q learning** to implement different structures: **fully centralized** and **fully decentralized**. Analyze how the agents communicate with each other in these different structures. And also analyze the convergence and variance of these structures. Finally, we will try **DQN** to see how deep neural networks performed on this problem set.

3. Implementation

3.1. Environment

- According to some basic tutorials online about how to build our own single agent environment, I have built a multi agent environment myself for training and testing.





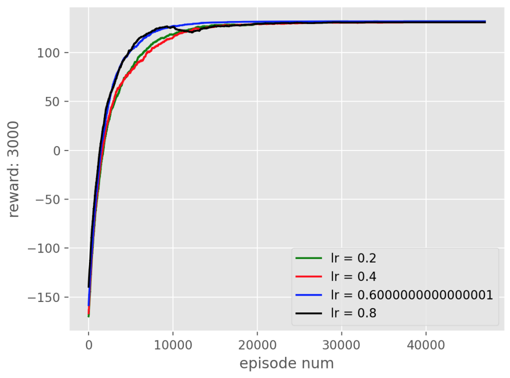
- **Environment introduction:** It's a <u>5*5 grid world</u> with some black areas representing the walls, as shown in <u>the picture on the left</u>:
  1) <u>State space:</u> there will be 20 states available for the agents to move around.
  2) <u>Agents and targets:</u> in this environment, there will be three agents (blue, red and green) representing the trucks and three targets (three orange areas) representing the charger stations. The agents will have specific start positions and could move in the environment, while the targets will have a fixed position and can not move in the environment. We fix the position of targets to leave more dimensions for agents.
  3) <u>Action space:</u>agents could take five actions {up, down, left, right, stay}.
  4) <u>Rewards:</u> the rewards defined inside the environment includes following rules:
     a) If one agent hits the target, get reward = 25.
     b) If two or more agents hit the same target, each will get punishment = -500.
     c) Each agent will get a punishment = -1 on each movement.
     d) Each agent will receive a time punishment = -1 if they did not hit the target in this movement.

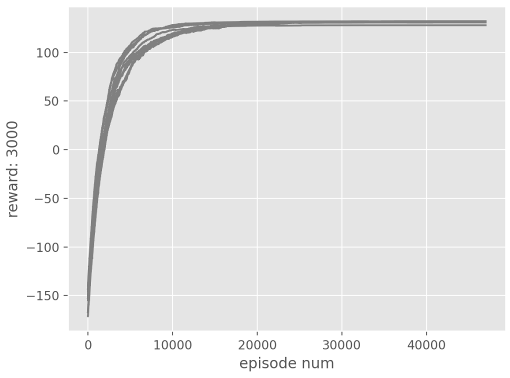
- **Best total reward and policy:** compute manually, we could get the best reward = 132, which means the blue agent will hit the upper orange in 2 steps (+50-2-2), the red agent will hit the right orange in 5 steps (+50-5-5), the green agent will hit the orange at bottom in 2 steps (+50-2-2). <u>The picture on the right</u> is where the agents should be in the end. We could check whether the algorithms find the best optimal policy by training its functions.

3.2. Training and Execution Structure With Action-Value Function:

3.2.1. Fully centralized

- **Brief introduction:** Fully centralized structure means that all the agents will share their information with each other. We should train and get the optimal joint policy by considering all the observations in time t.
- **Algorithm:**
  1) Parameters: episodes = 50000, T = 10 (run 10 times); ϵ = 0.9, decay = 0.9998 (ϵ −policy); γ =0.95 (discount); α = [0.2, 0.4, 0.6, 0.8] (learning rate);
  2) Start with any Q table:
  $$Q_0[(pos1, pos2, pos3)][act1][act2][act3]$$
  where 'pos' and 'act' means the positions and the actions of agents.
  3) In each iteration, update the Q table as following:
  $$TD \ = total\,rewards \ + \gamma_t sup\{Q_t[s_{t+1}]\} - Q_t[s_t][a_t]$$
  $$Q_{t+1}[s_t][a_t] = Q_t[s_t][a_t] + \alpha_t TD$$
  here s_t and a_t represent all the state space and action space.
  4) Stop when we reach the max iterations.
  5) Then we could get an optimized Q table and use it to find the best policy for all the trucks.
- **Results and analysis:**





  - **First, train with different learning rates, as shown in the left picture:**
    1) We could get a conclusion roughly: with higher learning rate, the model will reach the best total rewards with fewer episodes, which means the algorithm will converge faster.
    2) When learning rate = 0.8, there is a small period of deterioration. Maybe if the learning rate is too high, the algorithm may jump out of the optimal convergence process and get a lower reward.
    3) According to that, we will choose the **learning rate=0.5**.
  - **Then, train with learning rate=0.5 and run 10 times, as in the right picture:**
    1) We get: **variance = 1.289**; **mean = 131.1**
    2) In every run time, the model could be **close to the best total reward 132.0**, but may not exactly reach this point. This may be because we choose the policy in some randomness and not all of the action combinations will be reached in the training process. Thus the Q table could not make sure that it can find the best policy every time.

3.2.2. Fully decentralized

- **Brief introduction:** Fully decentralized means that all the agents only know their own observations and rewards from the environment. We need to create a Q table for each agent, and train them separately using their own rewards.
- **Algorithm:**
  1) Parameters: episodes = 50000, T = 10 (run 10 times); ϵ = 0.9, decay = 0.9998 (ϵ −policy); γ =0.95 (discount); α = [0.2, 0.4, 0.6, 0.8] (learning rate);
  2) Start with any Q table for each agent
  3) In each iteration, update each Q table as following:

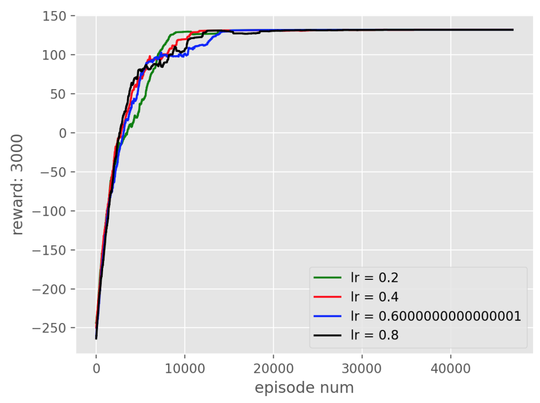
     $$Q^1_0[pos1][act1],\ Q^2_0[pos2][act2],\ Q^3_0[pos3][act3]$$

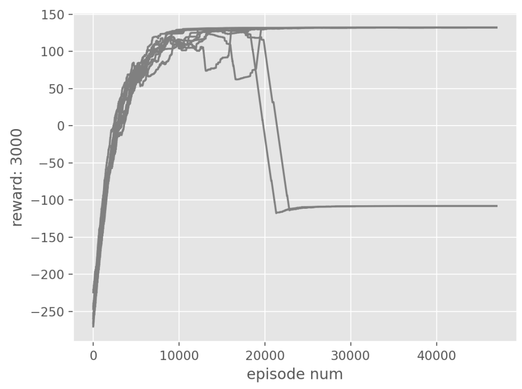
     where Q^i_t means the Q table for agent i at time t.
  4) In each iteration, update each Q table as following:
  5) Stop when we reach the max iterations.

     $$TD^i = reward\ of\ i + \gamma_t sup\{Q^i_t[s^i_{t+1}]\} - Q^i_t[s^i_t][a^i_t]$$

     $$Q^i_{t+1}[s^i_t][a^i_t] = Q^i_t[s^i_t][a^i_t] + \alpha_t TD^i$$

     doing these updating for each agent's Q table.
  6) Stop when we reach the max iterations.
  7) Then we will get three Q tables, that is each agent will have a Q table for their own. And use them to find the best policy for each truck.
- **Results:**





  - **First, train the model with different learning rates as in the left picture:**
    1) We could also get the rough conclusion according to the stage of rapid rise: with higher learning rate, the model will converge faster.
    2) For all the training with the learning rate larger than 0.2, there are more **zigzag lines** compared with fully centralized structure. This may be because we update the Q table separately. If we update them too quickly, the actions of other agents will be more unpredictable, which may lead to waste of steps and time and get lower total rewards.
    3) But all the training process could have the probability to **exactly reach the best total rewards 132.0**:
       a) This may be because each agent will receive rewards from the environment, which helps for cooperation. For example, if two agents hit the same target, the first one will get reward but the

second will get punishment. Thus the second one will be less likely to reach this position in the future.

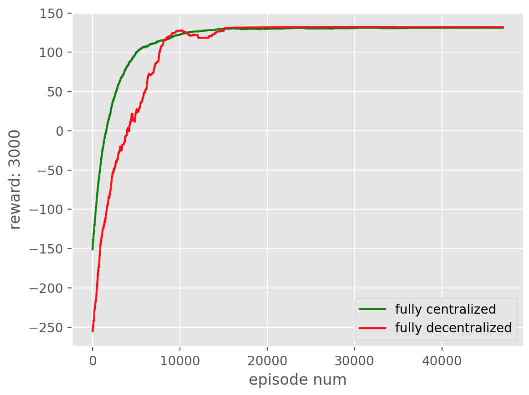
        b) With their own Q tables, the agents could find the fast path to the target that has more reward regardless of the detailed policy of other agents.

    4) According to this, we will choose the **learning rate=0.2**.

- **Then, train with learning rate=0.2 and run 10 times as in the right picture:**
    1) We get: **variance = 9216**; **mean = 89**
    2) This huge variance and low mean is because in some training, the model could reach convergence; while in others, it will **be trapped in a bad reward value, which is always around -108**. Even when we change the learning rate to 0.1 and slow down the decay of epsilon, this bad performance will happen occasionally in 10 run times.
        a) When we look into the agents' movement, we find that the middle agent (red one) will just stay and do not move in the whole process (-200), while the other two agents (blue and green) will reach the upper and bottom target (each +50-4). And thus will get a total reward of -108.
        b) Why will the red one just stay and not move at all? I think the reason may be that if the agent randomly chooses the 'stay' action, then it will be trapped in the 'stay' situation. Here the agent only has the Q table about its own actions and observations, thus when it is trapped in one state, it can only stay there forever. While in the fully centralized structure, the Q table includes all the observations and actions, thus when one agent chooses a bad action, there may be other combinations of actions that could guide the agent out.

3.2.3. Comparison of different structures with action-value function

- Run fully centralized and fully decentralized structures with learning rate = 0.5, and below is one of the result during running:

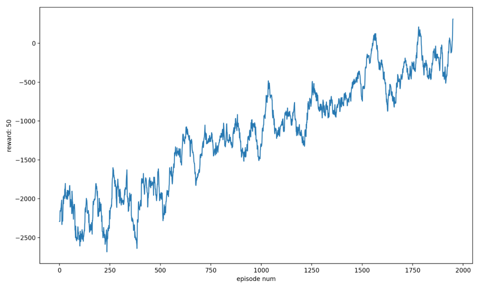


    - **Converge speed:** fully centralized structure will converge faster than fully decentralized structure. And the line of fully centralized structure is smoother.
    - **Converge or not:** fully decentralized structure could not guarantee that the best total rewards could be reached every time. While the fully centralized

structure could reach the best total rewards basically every time.
    - **Reach -108 or not:** we will get the bad performance from fully decentralized structure sometime during running, while we have not seen that bad performance from fully centralized structure.
    - **Variance**: according to the analysis above, fully centralized training will have a lower variance than fully decentralized training.
- **Application situation notation:**
    - The fully centralized structure could work even when the start states of agents are random. Because it has the Q table to record the probability of each action combination at each state combination.
    - The fully decentralized structure could not converge when the start states of agents are random, and could only be trained and converged when the start states are fixed. Because with different start states, the agents should take different actions according to their relative positions. But here the agents only have their own information, thus the Q table for one start states may not be suitable for another different start states.

3.3. DQN

- **Brief introduction:** Deep Q Network will use neural networks to do the function approximation. Here we try DQN based on value iteration algorithm, that is we will use one neural network to approximate the value function (Q table) of the model.
- **Algorithm:**
    1) Parameters: episodes=4000; ϵ=0.9, decay=0.9975; γ =0.99; batch size=32
    2) Start with any weights in the neural network:
        a) neural network has 5 dense layers; input is the state space in size (3,); output is the action space in size (5*5*5,)
        b) learning rate = 0.001
    3) For each episode:
        a) reset the environment
        b) take actions with epsilon randomness
        c) update the environment, and get (rewards, next_state, done).
        d) save [state, action, rewards, next_state, done] into the replay memory
        e) train the neural network with batches sampled randomly from the replay memory
        f) if not done, continue; if done, update epsilon and start a new episode
- Results:



- ○ Compared with action-value function based algorithms, the DQN has more parameters to be optimized, thus it needs more time. But the size of the problem will not increase exponentially in higher dimensional problems.

4. Conclusion and Future work:

- **In this report**:
  - We have explored the performance of fully centralized and fully decentralized structures on the simplified version of finding proper chargers for trucks. The fully centralized structure has a lower variance than the fully decentralized, thus we prefer to choose the fully centralized structure in the future research.
  - We have searched into the performance of fully decentralized structure. And found that the rewards feedback from the environment could provide some information about how to cooperate with each other. And we have also encountered a bad performance that could happen during the training of fully decentralized structure, and analyzed its detailed movement situation.
  - We have tried the DQN to approximate the value function. With the restriction of computers and time, we could not experiment more. But we have seen a trend of optimized total reward.
  - In conclusion, this report is a fundamental research of the basic functions for multi agent reinforcement learning, which could help us understand how the agents communicate and cooperate with each other. And it will be helpful for our future research with other algorithms.
- **In further research:**
  - We want to improve the environment with a bigger grid world and more agents and targets. We also want to add the charging information like start and end charging time into the environment.
  - With the growing of the world scope, we want to try to use DQN or other deep multi agent reinforcement learning algorithms to solve the problem with higher dimensions.

5. References

[1] Pablo Hernandez-Leal, Bilal Kartal and Matthew E. Taylor, *A Survey and Critique of Multiagent Deep Reinforcement Learning*, 2019

[2] Jakob N Foerster, *Deep multi-agent reinforcement learning*, 2018

[3] Ryan Lowe, Yi Wu, *Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments*, 2020

[4] Gonzalo Neto, *From Single-Agent to Multi-Agent Reinforcement Learning: Foundational Concepts and Methods*

[5] Marco Wiering, *Multi-Agent Reinforcement Learning for Traffic Light Control*

[6] Shai Shalev-Shwartz, Shaked Shammah, Ammon Shashua, *Multi-Agent Reinforcement Learning for Traffic Light Control,* 2016.10